2

SUBMITTED TO: 13TH INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, MAY 25-28, 1993, PITTSBURGH, PENNSYLVANIA

Lazy Checkpoint Coordination for Bounding Rollback Propagation[1]

Yi-Min Wang and W. Kent Fuchs

Primary contact: Yi-Min Wang

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
1308 West Main Street
University of Illinois
Urbana, IL 61801

E-mail: ymwang@crhc.uiuc.edu
Phone: (217) 244-7161
FAX: (217) 244-5686

Abstract

Independent checkpointing allows maximum process autonomy but suffers from potential domino effects. Coordinated checkpointing eliminates the domino effect by sacrificing a certain degree of process autonomy. In this paper, we propose the technique of lazy checkpoint coordination which preserves process autonomy while employing communication-induced checkpoint coordination for bounding rollback propagation. The introduction of the notion of laziness allows a flexible trade-off between the cost for checkpoint coordination and the average rollback distance. Worst-case overhead analysis provides a means for estimating the extra checkpoint overhead. Communication trace-driven simulation for several parallel programs is used to evaluate the benefits of the proposed scheme for real applications.

Key words: fault tolerance, independent checkpointing, checkpoint coordination, rollback recovery

[1]*Acknowledgement:* This research was supported in part by the National Aeronautics and Space Administration (NASA) under Grant NASA NAG 1-613, in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS), and in part by the Department of the Navy and managed by the Office of the Chief of Naval Research under Contract N00014-91-J-1283.

1 Introduction

Independent (or uncoordinated) checkpointing [1–3] for parallel and distributed systems allows maximum process autonomy and independent design of recovery capability for each process. However, since the rollback of a message sender requires the *sympathetic rollback* [4] of the receiver, the domino effect [5] is in general possible unless certain mechanisms are incorporated into the checkpointing and recovery protocol to guarantee recovery line [6] progression. Existing techniques for achieving domino-free rollback recovery can be classified into two primary categories [7]. The first category can be called the *minimum sympathetic rollback approach* in which either the rollback of a process will never undo any messages sent or the receiver of an undone message M will try to roll back to the state *immediately before* receiving M. Wu and Fuchs [8] insert a checkpoint immediately after each message is sent so that no sympathetic rollback is necessary for any failure. Kim et al. [9, 10] and Venkatesh et al. [11] employ dependency tracking and insert extra checkpoints before processing any messages that result in new dependency. The *state-interval based approach* [12–21] models the program execution as consisting of a number of *state intervals*, each started by processing a new message. Message logging in addition to checkpointing is employed to effectively insert an "checkpoint" (in the optimized form of a message log) before each message receipt.

The second category can be called the *bounded rollback propagation approach*. Corresponding checkpoints (based on the ordinal numbers) on different processes are required to coordinate with each other in order to form a recovery line to bound the possible rollback propagation. Usually, whenever a checkpoint is initiated by one process, all other processes are informed and required to take appropriate checkpoints to guarantee the resulting set of checkpoints is consistent [22–27]. The number of processes required to participate in each checkpointing session can be reduced by monitoring the recent message exchanging history [28]. For systems with clock synchronization and/or bounded message transmission delay, the cost for checkpoint coordination can be further reduced [29–32].

We will use the term *eager checkpoint coordination* for the coordination action performed when checkpoints are *initiated* (as described above). In contrast, processes in a system with *lazy checkpoint coordination* only coordinate their corresponding checkpoints when the message communication indicates a violation of checkpoint consistency[2]. Briatico et al. [35] force the receiver of a message M to take a checkpoint before processing M if the sender's checkpoint ordinal number tagged on M is greater than that of the receiver. Checkpoints with the same ordinal numbers are therefore always guaranteed to be consistent. However, the run-time overhead may be prohibitively high due to the possibly excessive number of extra induced checkpoints. In this paper, we generalize the concept of communication-induced checkpoint coordination by introducing the notion of *laziness* Z as a measure of the frequency for performing coordination. Only corresponding checkpoints with ordinal numbers nZ, where n is an integer, are required to be consistent with each other and form the recovery line for bounding rollback propagation. Overhead analysis and experimental evaluation show that lazy checkpoint coordination can significantly reduce the number of extra checkpoints and offer a flexible trade-off between run-time overhead versus average rollback distance.

The paper is organized as follows. Section 2 describes the system model and the checkpointing and recovery protocol; Section 3 gives the motivation and the algorithm for lazy checkpoint coordination; Worst-case overhead analysis is presented in Section 4 and the trace-driven simulation results for several parallel programs are discussed in Section 5.

2 Checkpointing and Rollback Recovery

The system considered in this paper consists of a number of concurrent processes for which all process communication is through message passing. Processes are assumed to run on fail-stop processors [36] and each processor is considered as an individual *recovery unit* [15]. We do not assume the piecewise deterministic execution model [20].

During normal execution, the state of each processor is periodically saved as a *checkpoint* on stable storage. Let $CP_{i,k}$ denote the kth checkpoint of processor p_i with $k \geq 0$ and $0 \leq i \leq N-1$, where N is the number of processors. A *checkpoint interval* is defined to be the time between two consecutive checkpoints on the same processor and the interval between $CP_{i,k}$ and $CP_{i,(k+1)}$

[2]The basic idea motivating the lazy checkpoint coordination is similar to the concepts behind the *lazy release consistency* in distributed shared memory [33] and the *lazy message cancellation* in optimistic distributed simulation systems [34].

is called the kth checkpoint interval. Each message is tagged with the current checkpoint ordinal number and the processor number of the sender. Each processor takes its checkpoint independently and updates the *direct dependency information table* (or *input table* [2]) as follows: if at least one message from the mth checkpoint interval of processor p_j has been processed during the previous checkpoint interval, the pair (j, m) is added to the table entry for the new checkpoint.

A centralized *garbage collection algorithm* [37] can be periodically invoked by any processor. First, the dependency information for all existing checkpoints is collected to construct the *checkpoint graph* [1] (Fig. 1(b)). All checkpoints corresponding to the vertices marked "X" in Fig. 1 (b) are determined to be garbage by the algorithm and can therefore be discarded.

When processor p_i initiates a rollback, it sends out a *rollback_initiating* message [2] to every other processor to request the up-to-date dependency information. Each surviving processor takes a *virtual checkpoint* (represented by the dotted vertex in Fig. 1 (c)) upon receiving the *rollback_initiating* message. After receiving the responses, p_i constructs the *extended checkpoint graph* [1] and executes the rollback propagation algorithm shown in Fig. 2 to determine the recovery line (the shaded vertices in Fig. 1 (c)). A *rollback_request* message is then broadcast to roll back each processor according to the recovery line (Fig. 1 (d)).

There are two primary checkpoint consistency situations. In Fig. 3(a), the checkpoints $CP_{i,k}$ and $CP_{j,m}$ are inconsistent because of the *orphan message* [31] M_a. In Fig. 3(b), $CP_{i,k}$ and $CP_{j,m}$ can become consistent if the *channel-state message* [24] M_b is properly recorded. In this paper, we assume either every message is synchronously logged[3] [12, 14] or an end-to-end transmission protocol can guarantee the redelivery of the lost channel-state messages [28]. Therefore, checkpoints like $CP_{i,k}$ and $CP_{j,m}$ in Fig. 3(b) are considered consistent.

3 Lazy Checkpoint Coordination

3.1 Motivation

We will refer to the checkpoints initiated independently by each processor as *basic checkpoints* and those triggered by the communication as *induced checkpoints*. Fig. 4(a) illustrates the situation

[3]Discussions on incorporating an asynchronous logging protocol into the independent checkpointing scheme described in this section can be found in [3].

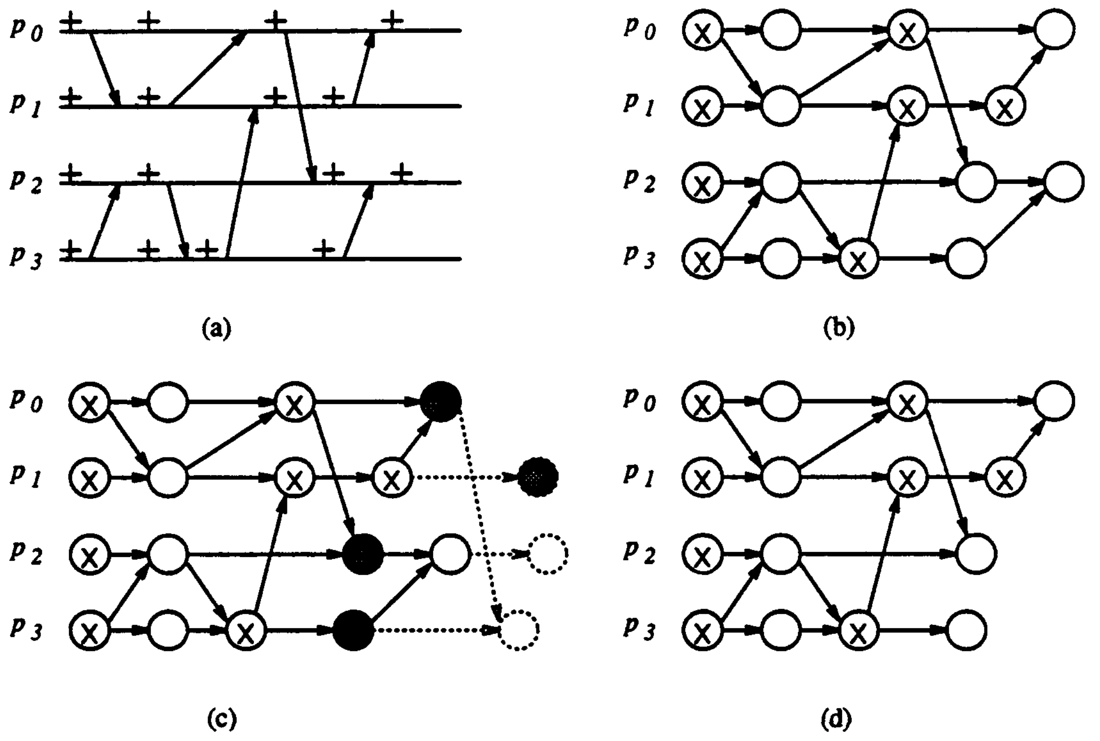


Figure 1: Checkpointing and rollback recovery (a) the checkpoint and communication pattern (b) checkpoint graph for garbage collection (c) extended checkpoint graph when p_0 initiates the rollback (d) checkpoint graph after recovery.

```
/* CP stands for checkpoint */
/* Initially, all the CPs are unmarked */
Include the latest CP of each processor in the root set;
Mark all CPs strictly reachable from any CP in the root set;
While (at least one CP in the root set is marked) {
    Replace each marked CP in the root set by the latest unmarked CP on the same
    processor;
    Mark all CPs strictly reachable from any CP in the root set;
}
The root set is the recovery line.
```

Figure 2: The rollback propagation algorithm.

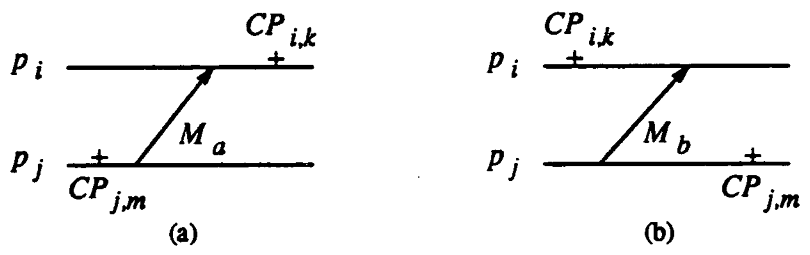


Figure 3: Checkpoint consistency (a) orphan message (b) channel-state message.

where the communication pattern renders most of the basic checkpoints useless for rollback recovery and the only recovery line is at the very beginning of the execution. A straightforward way of avoiding such unbounded rollback propagation is to perform eager checkpoint coordination as shown in Fig 4(b). Whenever a processor initiates a basic checkpoint, *coordination messages* (dotted arrows) are broadcast to all other processors to request the cooperation in making a consistent set of checkpoints [23]. Let B be the total number of basic checkpoints and I be the total number of induced checkpoints. We define the *induction ratio* $\mathcal{R}$ as

$$\mathcal{R} = \frac{I}{B} \tag{1}$$

which is a measure of the overhead for performing communication-induced checkpoint coordination. Clearly, eager checkpoint coordination has $\mathcal{R} = N - 1$ and will result in large run-time overhead when N is large. In addition, the $N - 1$ coordination messages per checkpoint session constitute another overhead.

The large overhead of eager checkpoint coordination results from its pessimistic nature. More specifically, when p_1 in Fig 4(b) initiates its first basic checkpoint $b_{1,1}$[4], it "pessimistically" assumes that messages like M_1 will exist in the future and cause $b_{1,1}$ to be inconsistent with its corresponding checkpoint $b_{0,1}$ on p_0. In order to guarantee $b_{1,1}$ belongs to a useful recovery line, p_1 "eagerly" requests p_0's cooperation at the time $b_{1,1}$ is initiated. In contrast, lazy checkpoint coordination adopts an optimistic approach by assuming that $b_{0,1}$ will be consistent with $b_{1,1}$. If the assumption turns out to be true, no explicit coordination is necessary. An extra checkpoint will be induced on p_0 only when the message M_1 indicates that the assumption has failed (Fig 4(c)). From another point of view, such a scheme "lazily" delays the broadcast of the coordination messages and implicitly

[4] $b_{i,k}$ denotes the kth basic checkpoint of p_i and $CP_{i,k}$ denotes the kth checkpoint of p_i.

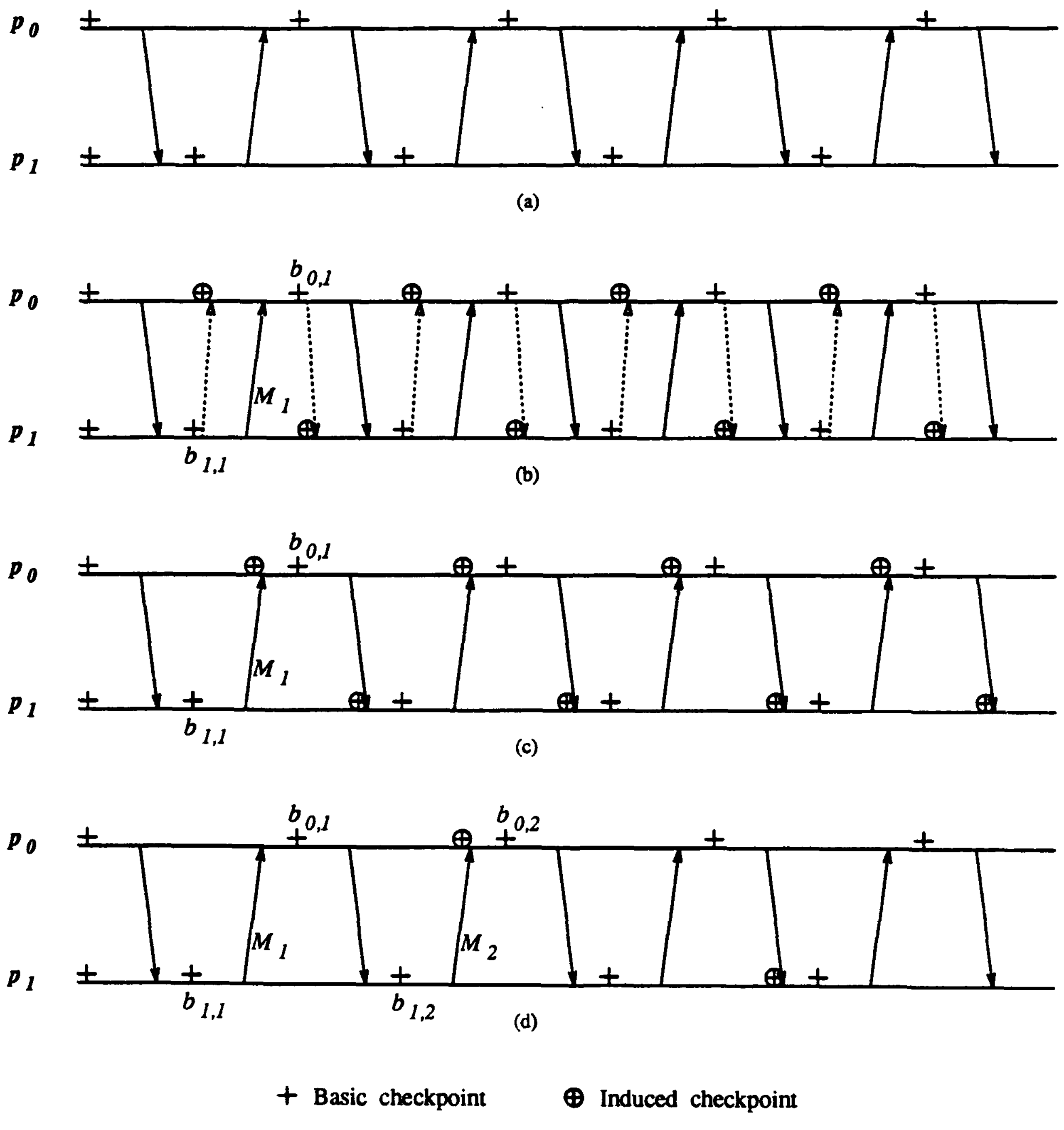


Figure 4: Communication-induced checkpointing (a) the checkpoint and communication pattern (b) eager checkpoint coordination (c) lazy checkpoint coordination with laziness = 1 (d) lazy checkpoint coordination with laziness = 2.

piggybacks them on future normal messages. Both checkpoint and message overhead can therefore be reduced.

However, given a basic checkpoint pattern, the number of induced checkpoints in the above scheme is determined by the communication pattern and is not otherwise controllable. In the worst case, the induction ratio $\mathcal{R}$ can still be $N-1$ as illustrated in Fig 4(c). In order to further reduce the overhead, we can perform even "lazier" coordination by only enforcing the consistency between checkpoints $CP_{0,nZ}$ and $CP_{1,nZ}$ where Z is called the *laziness* and n is an integer. Fig 4(d) shows the case with $Z = 2$. No checkpoint is induced until the message M_2 indicates the inconsistency between $b_{1,2}$ and $b_{0,2}$. The number of induced checkpoint can be reduced from 8 (Fig 4(c) with $Z = 1$) to 2 at the cost of potentially larger rollback distance. It becomes clear that lazy checkpoint coordination can provide a trade-off between the checkpointing overhead and average rollback distance.

3.2 The Protocol

Our approach is to incorporate the lazy checkpoint coordination into the independent checkpointing scheme as a mechanism for bounding rollback propagation. Therefore, the checkpointing and rollback recovery protocol can be built on top of the one described in Section 2. During normal execution, each processor still takes its basic checkpoints independently. The laziness Z is a predetermined-determined system parameter known to all processors. Suppose a processor p_j with current checkpoint ordinal number r is about to process a message M with sender p_i's ordinal number s. If p_j detects the following condition to be true

$$l = \lfloor s/Z \rfloor > \lfloor r/Z \rfloor,$$

it realizes that $CP_{i,lZ}$ and $CP_{j,lZ}$ will be inconsistent unless an extra checkpoint is induced before M is processed. We describe a possible implementation as follows. Each processor p_j maintains a variable V which is initialized to be Z and incremented by Z each time $CP_{j,nZ}$ is taken. Before p_j processes a message M with $s \geq V$, it is forced to take the checkpoint $CP_{j,lZ}$ and update its ordinal number counter to lZ. In other words, if M was sent after $CP_{i,lZ}$ was taken, it must be processed by p_j after $CP_{j,lZ}$ is induced. Notice that all checkpoints $CP_{j,m}$ with $r < m < lZ$ become *dummy checkpoints* which overlap with $CP_{j,lZ}$.

In addition to the centralized garbage collection algorithm [37], a simple distributed algorithm can also be used for low-cost garbage collection. The basic idea is that if the current checkpoint ordinal number of every processor has exceeded nZ, all the checkpoints $CP_{j,m}$ with $m < nZ$ becomes obsolete with respect to the recovery line consisting of $\{CP_{i,nZ} : 0 \leq i \leq N-1\}$ and therefore can be discarded. Each processor p_j needs to maintain an array *CP_progress*$[N]$ which records the highest ordinal number for every other processor known to p_j based on the information included in each message. More efficient garbage collection can be achieved by piggybacking the *CP_progress*$[N]$ array on the normal messages periodically in order to maintain the "transitive" knowledge of checkpointing progress of each processor [38].

Although the set of checkpoints $\{CP_{i,nZ} : 0 \leq i \leq N-1\}$ always forms a recovery line, the two-phase recovery procedure described in Section 2 should still be used to search for the most recent recovery line in order to minimize the number of rolled-back processors and the rollback distance. One possible optimization is that the dependency information corresponding to the garbage checkpoints as determined based on the *CP_progress*$[N]$ array needs not be collected, thus reducing the size of the responses to the *rollback_initiating* message and the time for constructing the checkpoint graph.

4 Overhead Analysis

Since the checkpoint overhead of the lazy checkpoint coordination scheme depends on the run-time dynamic communication pattern, it is important to analyze and estimate the potential extra overhead resulting from the induced checkpoints. We will first show that, without any constraints on the relative checkpointing progress of each processor, the worst-case induction ratio is $(N-1)/Z$. While under certain conditions which are typically met by real applications, the upper bound on the induction ratio can be shown to be independent of N.

4.1 Worst-Case Analysis

Our approach to worst-case analysis consists of two steps. First, given any fixed basic checkpoint pattern, we construct the worst-case communication pattern. Secondly, given any system with N processors, we derive the worst-case induction ratio as a function of N and the laziness Z.

In this section, we assume each checkpoint $CP_{i,k}$ is associated with a global time stamp $t(CP_{i,k})$[5]. For any checkpoint and communication pattern $\mathcal{P}$, define $CP^{\mathcal{P}}_{*,nZ} = CP^{\mathcal{P}}_{i,nZ}$[6] if $t(CP^{\mathcal{P}}_{i,nZ}) \leq t(CP^{\mathcal{P}}_{j,nZ})$ for all $0 \leq j \leq N-1$, i.e., $CP^{\mathcal{P}}_{*,nZ}$ denotes the earliest checkpoint $\#nZ$ among all processors. Given any basic checkpoint pattern and the laziness Z, we construct the communication pattern $\mathcal{P}_0$ as follows. Suppose $CP^{\mathcal{P}_0}_{*,nZ} = CP^{\mathcal{P}_0}_{i,nZ}$. Then p_i sends a message to every other processor and induces $CP^{\mathcal{P}_0}_{j,nZ}$ with $t(CP^{\mathcal{P}_0}_{j,nZ}) \approx t(CP^{\mathcal{P}_0}_{i,nZ})$ on processor p_j. Fig. 5(a) shows an example of $\mathcal{P}_0$ with $Z = 2$. We will call the interval between $t(CP^{\mathcal{P}_0}_{*,(n-1)Z})$ and $t(CP^{\mathcal{P}_0}_{*,nZ})$ the *induction session* $\#n$ which includes all the induced checkpoints $CP^{\mathcal{P}_0}_{j,nZ}$. The following lemma will be used to prove $\mathcal{P}_0$ is the worst-case communication pattern in terms of the induction ratio.

LEMMA 1 *Given a basic checkpoint pattern, we have* $t(CP^{\mathcal{P}_0}_{*,nZ}) \leq t(CP^{\mathcal{P}}_{*,nZ})$ *for arbitrary communication pattern* $\mathcal{P}$ *and any positive integer* n.

Proof. The proof is given by induction on n. Since there can not be any induced checkpoint before $t(CP^{\mathcal{P}}_{*,Z})$ for any $\mathcal{P}$, $t(CP^{\mathcal{P}}_{*,Z})$ only depends on the progress of taking basic checkpoints. Therefore, $t(CP^{\mathcal{P}_0}_{*,Z}) = t(CP^{\mathcal{P}}_{*,Z})$ and the case $n = 1$ is true. For the case $n = k$, suppose $CP^{\mathcal{P}}_{*,kZ} = CP^{\mathcal{P}}_{i,kZ}$. All the Z checkpoints $CP^{\mathcal{P}}_{i,l}$ with $(k-1)Z < l \leq kZ$ must be basic checkpoints because they can not be induced before $t(CP^{\mathcal{P}}_{*,kZ})$. Also, $t(CP^{\mathcal{P}}_{*,(k-1)Z}) \leq t(CP^{\mathcal{P}}_{i,(k-1)Z}) \leq t(CP^{\mathcal{P}}_{i,l}) \leq t(CP^{\mathcal{P}}_{i,kZ})$ by definition. Suppose the case $n = k-1$ is true, i.e., $t(CP^{\mathcal{P}_0}_{*,(k-1)Z}) \leq t(CP^{\mathcal{P}}_{*,(k-1)Z})$. We then have $CP^{\mathcal{P}}_{i,kZ} = CP^{\mathcal{P}_0}_{i,q}$ where $q \geq kZ$ because $t(CP^{\mathcal{P}_0}_{i,(k-1)Z}) \approx t(CP^{\mathcal{P}_0}_{*,(k-1)Z})$ by construction and there are at least Z basic checkpoints of p_i, i.e., the $CP^{\mathcal{P}}_{i,l}$'s, between $t(CP^{\mathcal{P}_0}_{i,(k-1)Z})$ and $t(CP^{\mathcal{P}}_{i,kZ})$. Finally,

$$t(CP^{\mathcal{P}_0}_{*,kZ}) \leq t(CP^{\mathcal{P}_0}_{i,kZ}) \leq t(CP^{\mathcal{P}_0}_{i,q}) = t(CP^{\mathcal{P}}_{i,kZ}) = t(CP^{\mathcal{P}}_{*,kZ})$$

and we have proved $t(CP^{\mathcal{P}_0}_{*,nZ}) \leq t(CP^{\mathcal{P}}_{*,nZ})$ for all positive integer n. □

LEMMA 2 *Given a basic checkpoint pattern,* $\mathcal{P}_0$ *is the worst-case communication pattern resulting in the largest induction ratio.*

[5]This is only for the purpose of presentation.

[6]We will use $CP^{\mathcal{P}}_{i,k}$ to denote the kth checkpoint of p_i in the checkpoint and communication pattern $\mathcal{P}$. When it is clear from the context that the basic checkpoint pattern is fixed, we also use the same notation for the communication pattern $\mathcal{P}$.

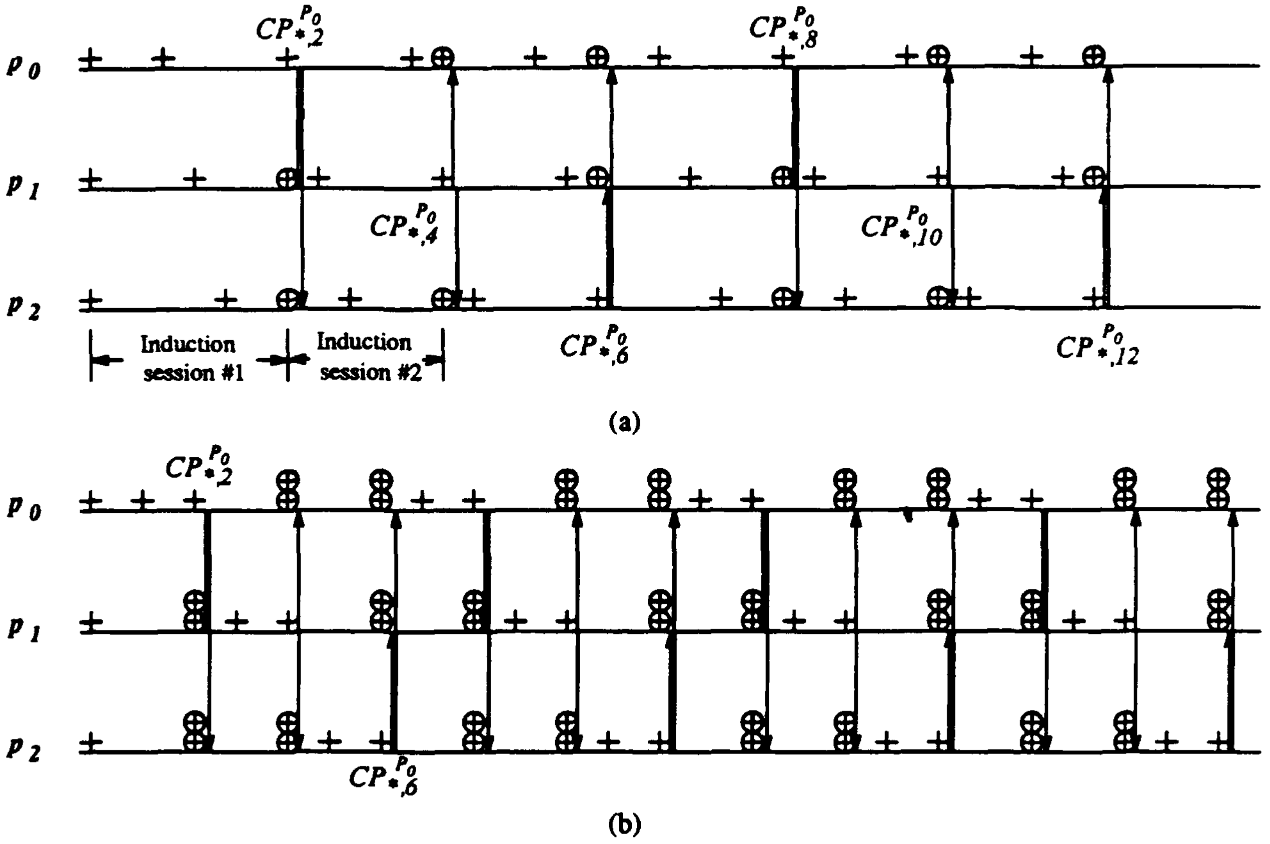


Figure 5: (a) Worst-case communication pattern (b) worst-case checkpoint and communication pattern.

Proof. Let $I_n^{\mathcal{P}}$ denote the total number of induced checkpoints with ordinal number nZ for the communication pattern $\mathcal{P}$, and let $q = max\{n : I_n^{\mathcal{P}} \neq 0\}$. be the maximum among n's such that $I_n^{\mathcal{P}} \neq 0$. Clearly, $I_n^{\mathcal{P}} \leq N - 1$. Since $t(CP^{\mathcal{P}_0}_{*,qZ}) \leq t(CP^{\mathcal{P}}_{*,qZ})$ by Lemma 1, the checkpoint and communication pattern with $\mathcal{P}_0$ must consist of at least q induction sessions. Let $I^{\mathcal{P}}$ denote the total number of induced checkpoints for $\mathcal{P}$, we then have

$$I^{\mathcal{P}_0} \geq \sum_{1 \leq n \leq q} I_n^{\mathcal{P}_0} = q \cdot (N-1) \geq \sum_{1 \leq n \leq q} I_n^{\mathcal{P}} = I^{\mathcal{P}}.$$

Finally, because the number of basic checkpoints is fixed by the given basic checkpoint pattern, $\mathcal{P}_0$ has the largest induction ratio among all possible communication patterns. □

Lemma 2 states that, for worst-case analysis of the induction ratio, we need only consider the communication pattern $\mathcal{P}_0$ for each basic checkpoint pattern. Because the induction sessions are well-defined in such patterns (as shown in Fig. 5), the derivations can be simplified.

THEOREM 1 *For any system with N processors and laziness Z, the induction ratio*

$$\mathcal{R} \leq \frac{N-1}{Z}.$$

Proof. For any basic checkpoint pattern with its corresponding $\mathcal{P}_0$ which results in L complete induction sessions, the number of induced checkpoints is $L \cdot (N-1)$. Let B_n denote the number of basic checkpoints within the induction session #n, we have $B_n \geq Z$ for all $1 \leq n \leq L$ because the Z checkpoints $CP_{i,l}^{\mathcal{P}_0}$ with $(n-1)Z < l \leq nZ$ can not be the induced checkpoints if $CP_{*,nZ}^{\mathcal{P}_0} = CP_{i,nZ}^{\mathcal{P}_0}$. Therefore, the induction ratio

$$R = \frac{L \cdot (N-1)}{\sum_{1 \leq n \leq L} B_n + B_{L+1}} \leq \frac{L \cdot (N-1)}{L \cdot Z} = \frac{N-1}{Z}.$$

□

Fig. 5(b) shows an example of the worst case for $N = 3$ and $Z = 2$. The stacked checkpoints indicate the fact that each dummy checkpoint $CP_{i,2n-1}^{\mathcal{P}_0}$ overlaps with the induced checkpoint $CP_{i,2n}^{\mathcal{P}_0}$. Since it takes exactly $Z = 2$ basic checkpoints to induce every $N - 1 = 2$ checkpoints, the induction ratio is $(N-1)/Z = 1$.

4.2 The Upper Bound under Constraints

The upper bound in Theorem 1 was derived under no constraints on the program behavior. Since it is of order $\mathbf{O}(N)$, the induction ratio may be unacceptably high for systems with large number of processors. However, a closer look at the two patterns in Fig. 5 reveals that the situation in (b) which results in the worst-case induction ratio is less likely to happen for real applications where the processors typically regularize their paces in taking basic checkpoints, as shown in (a). For example in Fig. 5(b), it is very likely for p_0 to take at least one basic checkpoint between $CP_{*,2}^{\mathcal{P}_0}$ and $CP_{*,6}^{\mathcal{P}_0}$. We can show that under the following constraints which are usually satisfied in real applications, the upper bound on the induction ratio is independent of N.

Constraint 1: Let Q denote the ratio of the maximum to the minimum length of the basic checkpoint interval. Although each processor is allowed to take its basic checkpoints at its own pace.

Q is typically bounded by a small constant $\hat{Q}$. For example, $\hat{Q}$ is 2 or 3 for our experiments described in the next section.

Constraint 2: Only the cases with $Z \geq 2$ will be considered for refined upper bounds because the worst case for $Z = 1$ is always achievable even when Q is small (see Fig. 4(c)).

Constraint 3: The applications employing checkpointing and rollback recovery are usually long-running jobs, which implies $Z \cdot L$ is quite large. (Recall L is the number of complete induction sessions with $\mathcal{P}_0$.) In particular, we assume $Z \cdot L \gg \lceil Q \rceil$.

THEOREM 2 *Under the above constraints, the induction ratio $\mathcal{R} < \lceil Q \rceil$.*

Proof. Again we only have to consider $\mathcal{P}_0$ for each basic checkpoint pattern for the worst case. Let M denote the smallest integer such that $M \cdot (Z-1) \geq Q$. Since $Z \geq 2$ by Constraint 2, we have $M \leq \lceil Q \rceil$. We define an ***M*-induction session** as consisting of M consecutive induction sessions. There are then $L_M = \lfloor L/M \rfloor$ complete M-induction sessions, each containing $M \cdot (N-1)$ induced checkpoints. We consider the following two cases.

(a) $N < M$: By Theorem 1,

$$\mathcal{R} \leq \frac{N-1}{Z} < N-1 < N < M \leq \lceil Q \rceil. \tag{2}$$

(b) $N \geq M$: First we consider the number of induced checkpoints I. If $Z \geq Q+1$, then $M = 1$ and $I = L \cdot (N-1)$. If $Z < Q+1$, $Z \cdot L \gg \lceil Q \rceil$ in Constraint 3 implies $L \gg \lceil Q \rceil$. Since $M \leq \lceil Q \rceil$, we have $L/M \gg 1$ and

$$I = L_M \cdot M \cdot (N-1) + \sum_{L_M \cdot M+1 \leq k \leq L} I_k \approx L_M \cdot M \cdot (N-1).$$

In either case, $I \approx L_M \cdot M \cdot (N-1)$.

Now consider the number of basic checkpoints B. For each induction session $\#n$, the processor p_i with $CP^{\mathcal{P}_0}_{i,nZ} = CP^{\mathcal{P}_0}_{*,nZ}$ must contribute Z basic checkpoints and therefore the length of each induction session is at least $Z-1$ basic checkpoint intervals. Within each M-induction session, at least $N-M$ processors do not have $CP^{\mathcal{P}_0}_{j,nZ} = CP^{\mathcal{P}_0}_{*,nZ}$ for any n. By the definition of Q, these $N - M$ processors must each contribute at least $\lfloor\frac{M\cdot(Z-1)}{Q}\rfloor$ basic checkpoints. Therefore,

$$B \geq L_M \cdot (M \cdot Z + (N - M) \cdot \lfloor\frac{M \cdot (Z-1)}{Q}\rfloor)$$

and

$$\mathcal{R} = \frac{I}{B} \leq \frac{M \cdot (N-1)}{M \cdot Z + (N-M) \cdot \lfloor\frac{M\cdot(Z-1)}{Q}\rfloor}. \tag{3}$$

Since $Z > 1$ and $\frac{M\cdot(Z-1)}{Q} \geq 1$ by definition, we have

$$\mathcal{R} < \frac{M \cdot (N-1)}{M + (N-M)} < M \leq \lceil Q \rceil. \tag{4}$$

□

Notice that Eqs. (3) and (4) are still valid if we replace M with any m such that $M \leq m \leq \lceil Q \rceil$. By combining Theorem 1, Eq. (2) and Eq. (3), we then define the refined upper bound, called the $Q - bound$, as follows.

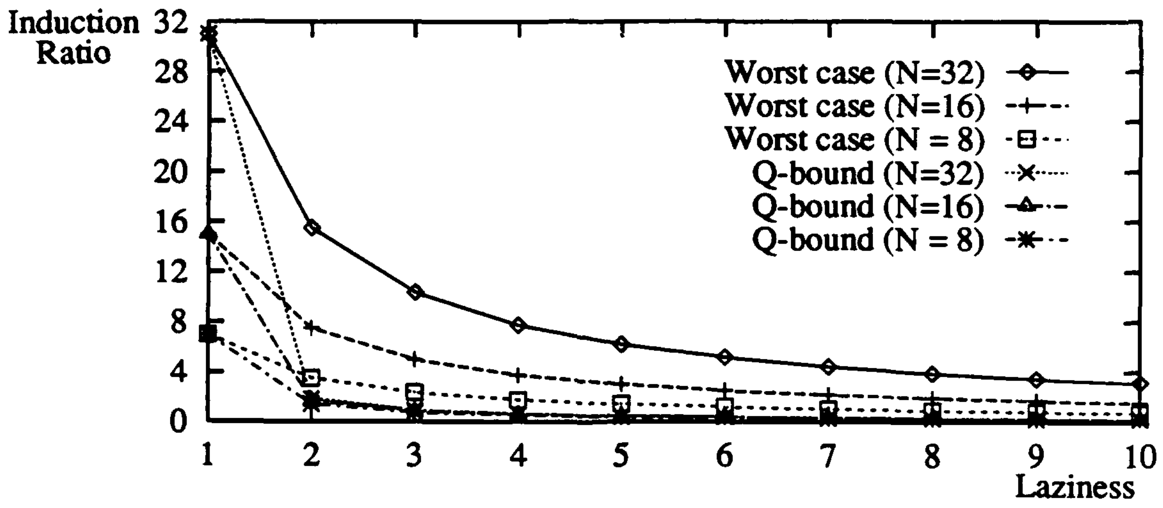
$$Q - bound = min_{M \leq m \leq \lceil Q \rceil}\{\frac{m \cdot (N-1)}{m \cdot Z + [N \geq m] \cdot ((N-m) \cdot \lfloor\frac{m\cdot(Z-1)}{Q}\rfloor)}\} \tag{5}$$

where $[N \geq m] = 1$ if $N \geq m$ is true and 0 otherwise.

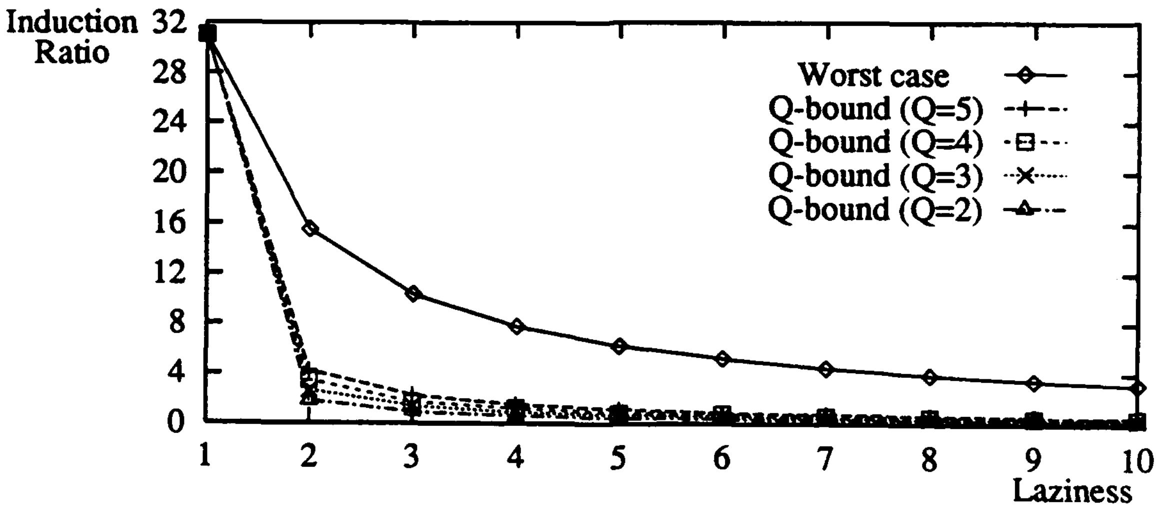
Fig. 6(a) compares the worst-case induction ratio with the $Q-bound$ where $Q = 2$ for $N = 8, 16$ and 32. While the worst-case ratio $(N-1)/Z$ clearly grows with N, the $Q - bound$ is relatively insensitive to N. Fig. 6(b) compares the worst-case induction ratio, which is equivalent to the $Q - bound$ with $Q = \infty$, with the $Q - bound$ where Q varies from 2 to 5. Since our purpose of introducing the $Q - bound$ is to estimate the induction ratio for real applications in advance, the insensitivity of the $Q-bound$ to the exact value of Q suggests that an approximate value of Q suffices for the estimation. Finally, notice that if Z is chosen to be at least $Q + 1$, we have $\mathcal{R} < M = 1$, i.e., the number of induced checkpoints will never exceed the number of basic checkpoints.

5 Experimental Results

Four parallel programs written in the *Chare Kernel* language are used for the communication trace-driven simulation. The Chare Kernel has been developed as a medium-grain, machine-



(b)



(a)

Figure 6: (a) Worst-case induction ratio and the $Q - bounds$ (Q=2) for various N (b) worst-case induction ratio ($N = 32$) and the $Q - bounds$ for various Q.

independent parallel language [39]. Programs written in the Chare Kernel language can run unchanged on both shared-memory and distributed-memory machines such as Encore Multimax, Sequent Symmetry, Intel iPSC/2 and i860 hypercubes and a network of Sun workstations. Program traces used in this paper are collected from an Multimax 510.

The four programs include two newly developed CAD applications, Test generation and Logic synthesis, and two search applications, Knight tour and N queen. The execution times are between 25 and 45 minutes (see Table 1). The total number of messages ranges from tens to hundreds of thousands. Our simulation uses the following scheme for inserting checkpoints. The predetermined minimum basic checkpoint interval is chosen to be 2 minutes. A variable *Next_CP_Time* is initialized to 2 minutes. Each processor checks its local clock after processing every 100 messages. If the clock time exceeds *Next_CP_Time*, a basic checkpoint is inserted and *Next_CP_Time* is incremented by 2 minutes. The resulting average basic checkpoint interval (CPI) for each program is listed in Table 1. Before processing each message, the processor also checks if an induced checkpoint and the corresponding update of the ordinal number counter are necessary, as described in Section 3. All reported numbers are averaged over five runs.

Table 1: Execution and checkpoint parameters of the Chare Kernel programs.

Programs	Test generation	Logic synthesis	Knight tour	N queen
Number of processors	8	6	8	6
Execution time (sec)	2,076	1,736	2,436	1,567
Number of messages	28,219	411,733	104,170	25,880
Average number of basic checkpoints per processor	12.6	11.8	18.0	10.5
Average basic CPI (sec)	158	140	132	139
Q	2.17	2.48	1.42	1.55
Under-2 percentage	99.6%	97.0%	100%	100%

We expect the variation of the basic checkpoint interval to be small because of the way it is maintained. In particular, we choose $Q = 2$ to estimate the induction ratio. The exact value of Q for each program is listed in Table 1. Although Q is slightly greater than 2 for the first two programs, the numbers listed in the row of "Under-2 percentage" shows that a very high percentage

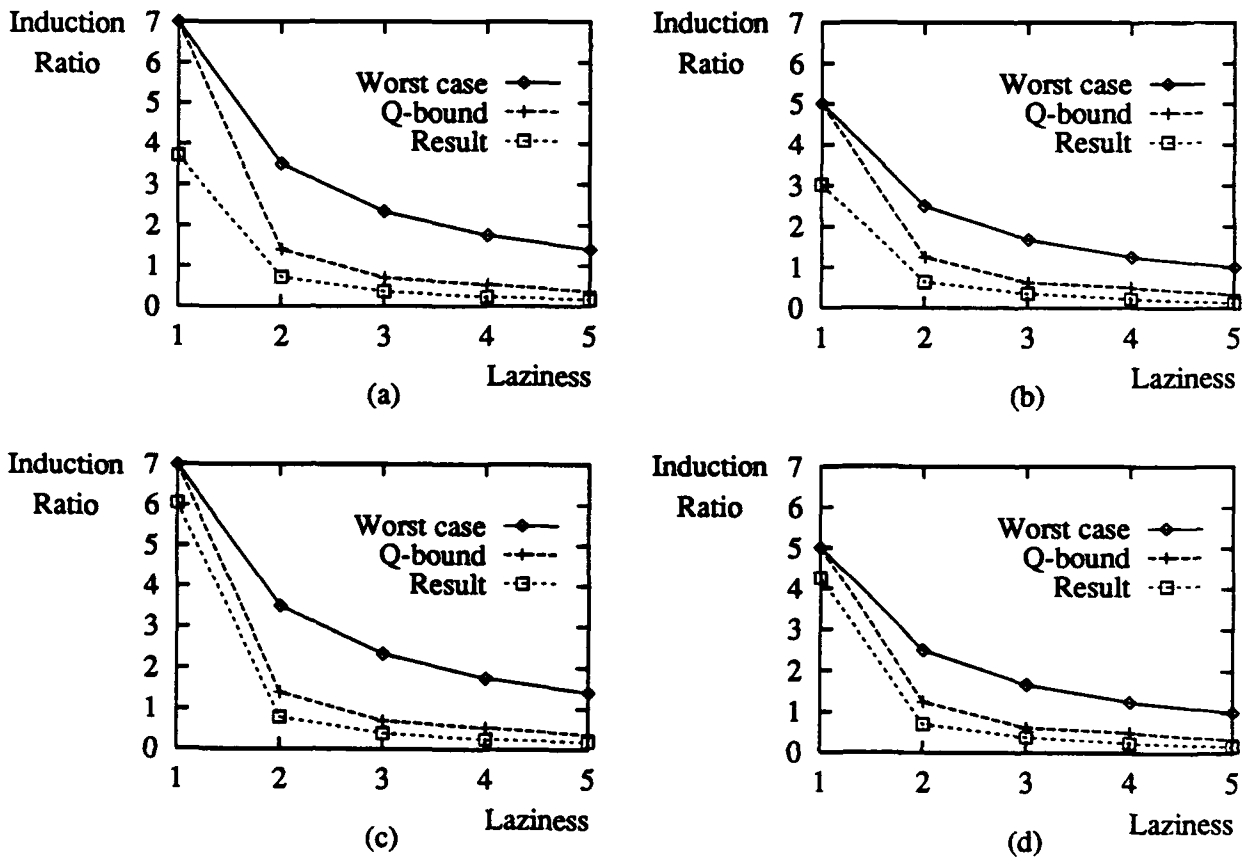


Figure 7: Checkpoint coordination overhead as a function of laziness (a) Test generation (b) Logic synthesis (c) Knight tour (d) N queen.

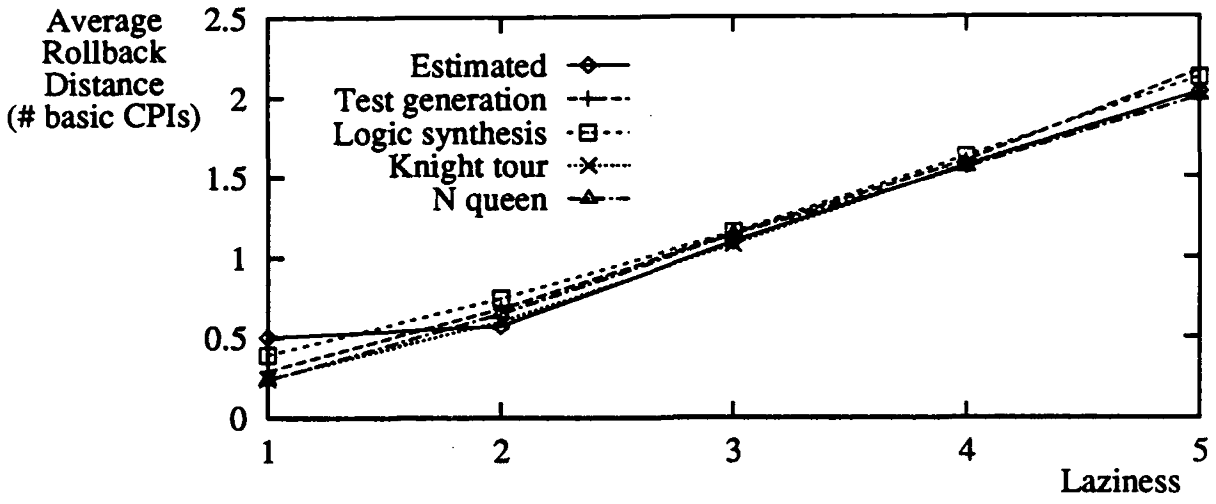


Figure 8: Average rollback distance as a function of the laziness.

of the basic checkpoint intervals are covered by $Q = 2$. Also, since the $Q - bound$ is insensitive to the exact value of Q, $Q = 2$ should suffice for our purpose. Fig. 7 plots the $Q - bounds$ against the worst-case and the actual induction ratios for the four programs. It is shown that the $Q - bound$ provides a good estimation of the induction ratio for real applications. The large difference in the ratio between $Z = 1$ and $Z \geq 2$ confirms that our generalization of the idea of communication-induced checkpoint coordination as described in [35] can significantly reduce the extra checkpoint overhead.

Fig. 8 gives the average rollback distances in terms of the number of average basic CPIs. The almost linear behavior can be explained as follows. Every N basic checkpoints $b_{i,k}$'s, $0 \leq i \leq N-1$, are taken at approximately the same time t_k. If any one of them, say $b_{j,k}$, is $CP_{*,nZ}$, then either $b_{i,k}$ is consistent with $b_{j,k}$ or $CP_{i,nZ}$ is induced shortly due to the relatively large number of messages. Hence, a recovery line is formed around t_k. For $Z = 1$, that means the average rollback distance is at most 0.5 basic CPI and the exact value will depend on the offset between $b_{i,k}$'s at run-time. For $Z \geq 2$, as long as some $CP_{i,nZ}$'s are induced before $b_{i,k}$'s are initiated, $b_{i,k}$'s become $CP_{i,nZ+1}$'s and one of $b_{i,k+(Z-1)}$'s will become $CP_{*,(n+1)Z}$ which means a new recovery line will very likely to exist around $t_{k+(Z-1)}$. Therefore, the average rollback distance is approximately $(Z-1)/2$ basic CPIs as shown by the curve named "Estimated" in Fig. 8. It becomes clear that Figs. 7 and 8 provide a flexible trade-off between run-time overhead and recovery efficiency.

6 Concluding Remarks

We have proposed the technique of lazy checkpoint coordination and incorporated it into the independent checkpointing protocol as a mechanism for bounding rollback propagation. The recovery line is guaranteed to move forward by performing communication-induced checkpoint coordination only when the predetermined consistency criterion is about to be violated. The notion of laziness was introduced to provide the trade-off between extra checkpoint overhead during normal execution versus the average rollback distance for recovery. Overhead analysis shows that the upper bound on the induction ratio, i.e., the number of induced checkpoints divided by the number of basic checkpoints, is related to the maximum ratio between the basic checkpoint intervals. Communication trace-driven simulation results for several parallel programs showed that our analysis can provide a good estimation for the induction ratio, and lazy checkpoint coordination can significantly reduce the extra checkpoint overhead for real applications.

Acknowledgement

The authors wish to express their sincere thanks to B. Ramkumar, K. De and P. Banerjee for their parallel programs and to L. V. Kalé for access to the Chare Kernel.

References

[1] K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic recovery schemes for distributed processes," in *Proc. IEEE 2nd Symp. on Reliability in Distributed Software and Database Systems*, pp. 124–130, 1981.

[2] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback for recovery - An optimistic approach," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 3–12, 1988.

[3] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 147–154, Oct. 1992.

[4] A. Lowry, J. R. Russell, and A. P. Goldberg, "Optimistic failure recovery for very large networks," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 66–75, 1991.

[5] B. Randell, "System structure for software fault tolerance," *IEEE Trans. on Software Engineering*, vol. SE-1, pp. 220–232, June 1975.

[6] P. A. Lee and T. Anderson, *Fault Tolerance Principles and Practice*. Springer-Verlag Wien, 1990.

[7] Y. M. Wang, A. Lowry, and W. K. Fuchs, "Checkpointing and rollback recovery for parallel and distributed systems: A survey." In preparation, 1992.

[8] K. L. Wu and W. K. Fuchs, "Recoverable distributed shared virtual memory," *IEEE Trans. on Computers*, vol. 39, pp. 460–469, Apr. 1990.

[9] K. H. Kim, J. H. You, and A. Abouelnaga, "A scheme for coordinated execution of independently designed recoverable distributed processes," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 130–135, 1986.

[10] K. H. Kim and J. H. You, "A highly decentralized implementation model for the Programmer-Transparent Coordination (PTC) scheme for cooperative recovery," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 282–289, 1990.

[11] K. Venkatesh, T. Radhakrishnan, and H. F. Li, "Optimal checkpointing and local recording for domino-free rollback recovery," *Information Processing Letters*, vol. 25, pp. 295–303, July 1987.

[12] A. Borg, J. Baumbach, and S. Glazer, "A message system supporting fault-tolerance," in *Proc. 9th ACM Symp. on Operating Systems Principles*, pp. 90–99, 1983.

[13] A. Borg, W. Blau, W. Graetsch, F. Herrmann, and W. Oberle, "Fault tolerance under UNIX," *ACM Trans. on Computer Systems*, vol. 7, pp. 1–24, Feb. 1989.

[14] M. L. Powell and D. L. Presotto, "Publishing: A reliable broadcast communication mechanism," in *Proc. 9th ACM Symp. on Operating Systems Principles*, pp. 100–109, 1983.

[15] R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. on Computer Systems*, vol. 3, pp. 204–226, Aug. 1985.

[16] A. P. Sistla and J. L. Welch, "Efficient distributed recovery using message logging," in *Proc. 8th ACM Symposium on Principles of Distributed Computing*, pp. 223–238, 1989.

[17] D. B. Johnson and W. Zwaenepoel, "Recovery in distributed systems using optimistic message logging and checkpointing," *J. of Algorithms*, vol. 11, pp. 462–491, 1990.

[18] T. T.-Y. Juang and S. Venkatesan, "Crash recovery with little overhead," in *Proc. IEEE Int'l Conf. on Distributed Computing Systems*, pp. 454–461, 1991.

[19] D. B. Johnson and W. Zwaenepoel, "Sender-based message logging," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 14–19, 1987.

[20] R. E. Strom, D. F. Bacon, and S. A. Yemini, "Volatile logging in n-fault-tolerant distributed systems," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 44–49, 1988.

[21] E. N. Elnozahy and W. Zwaenepoel, "Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit," *IEEE Trans. on Computers*, vol. 41, pp. 526–531, May 1992.

[22] Y. Tamir and C. H. Sequin, "Error recovery in multicomputers using global checkpoints," in *Proc. Int'l Conf. on Parallel Processing*, pp. 32–41, 1984.

[23] K. G. Shin and Y.-H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. on Software Engineering*, vol. 10, no. 6, pp. 692–700, 1984.

[24] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. on Computer Systems*, vol. 3, pp. 63–75, Feb. 1985.

[25] T. H. Lai and T. H. Yang, "On distributed snapshots," *Information Processing Letters*, vol. 25, pp. 153–158, May 1987.

[26] K. Li, J. F. Naughton, and J. S. Plank, "Checkpointing multicomputer applications," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 2–11, 1991.

[27] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The performance of consistent checkpointing," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 39–47, Oct. 1992.

[28] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. on Software Engineering*, vol. SE-13, pp. 23–31, Jan. 1987.

[29] P. Ramanathan and K. G. Shin, "Checkpointing and rollback recovery in a distributed system using common time base," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 13–21, 1988.

[30] F. Cristian and F. Jahanian, "A timestamp-based checkpointing protocol for long-lived distributed computations," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 12–20, 1991.

[31] Z. Tong, R. Y. Kain, and W. T. Tsai, "Rollback recovery in distributed systems using loosely synchronized clocks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, pp. 246–251, Mar. 1992.

[32] J. Long and W. K. Fuchs, "An evolutionary approach to coordinated checkpointing." to be submitted to *IEEE Trans. on Parallel and Distributed Systems*, 1992.

[33] P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy release consistency for software distributed shared memory," in *Proc. Int'l Symp. on Computer Architecture*, pp. 13–21, 1992.

[34] A. Gafni, "Rollback mechanisms for optimistic distributed simulation systems," in *Proc. SCS Multiconference on Distributed Simulation*, pp. 61–67, July 1988.

[35] D. Briatico, A. Ciuffoletti, and L. Simoncini, "A distributed domino-effect free recovery algorithm," in *Proc. IEEE 4th Symp. on Reliability in Distributed Software and Database Systems*, pp. 207–215, 1984.

[36] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. on Computer Systems*, vol. 1, pp. 222–238, Aug. 1983.

[37] Y. M. Wang, P. Y. Chung, I. J. Lin, and W. K. Fuchs, "Checkpoint space reclamation for independent checkpointing in message-passing systems." Tech. Rep. CRHC-92-06, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. Submitted to *IEEE Trans. on Parallel and Distributed Systems*, 1992.

[38] Y. M. Wang and W. K. Fuchs, "Scheduling message processing for reducing rollback propagation," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 204–211, July 1992.

[39] W. Shu and L. V. Kalé, "Chare kernel - A runtime support system for parallel computations," *J. Parallel and Distributed Computing*, vol. 11, pp. 198–211, 1991.